Enhanced Programming Interfaces for MPI

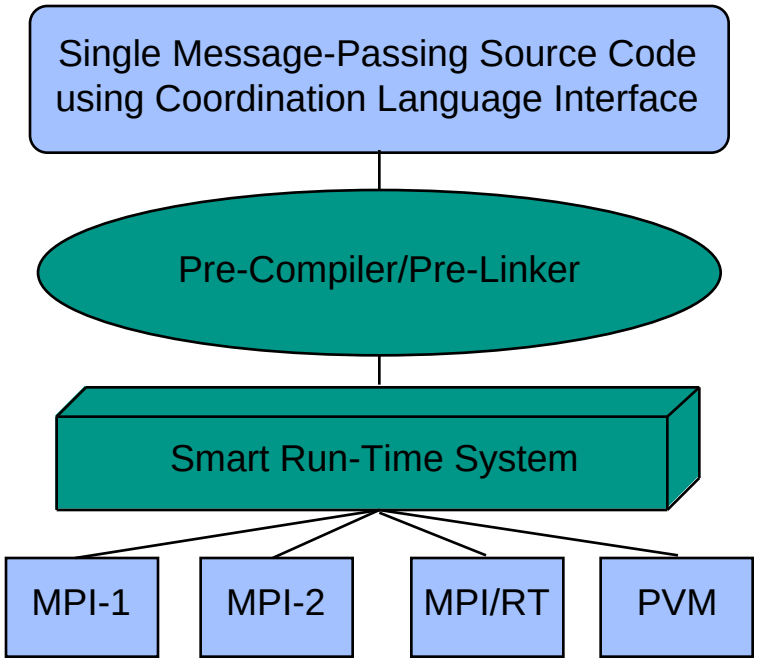
Andy Sherman (sherman@sca.com)
Nick Carriero (carriero@sca.com)

Scientific Computing Associates, Inc.
New Haven, Connecticut
(203) 777-7442

March 1997

SCIENTIFIC

Compiler Support for Message-Passing Systems



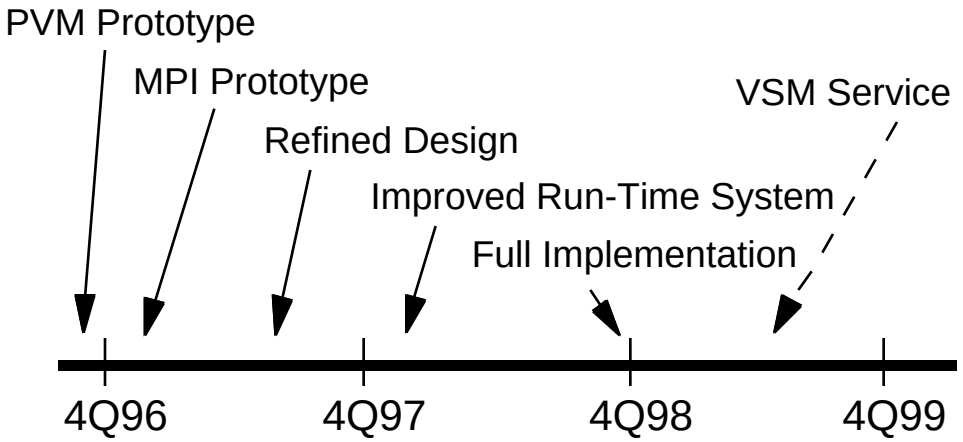
New Ideas

- Language-level support for message-passing systems
- Smart (dynamic & adaptive) run-time systems for message-passing
- Combining virtual shared memory with message-passing systems

Impact

- Substantial reductions in time & cost to develop new applications
- Simpler, shorter codes
- Extended lifetimes for applications
- Improved application performance
- Better in-field monitoring

Schedule



SCIENTIFIC

Project Objectives

- **Research on techniques to make parallel and distributed programming easier and more effective:**
  - Simplified API for message-passing systems
  - Reduced code size
  - Improved error detection and reporting
  - Enhanced portability
  - Flexible debugging/monitoring interface
  - Support for general algorithmic paradigms
- **Prototype software tools**
- **Applications**

Enhanced MPI Example

Here is an MPI code fragment:

```
/* ... Several MPI_Pack_size calls ... */
buf = malloc((unsigned) bsize);
pos = 0;
MPI_Pack(&nproc, 1, MPI_INT, buf, bsize, &pos, comm );
MPI_Pack(pdata, nproc, MPI_INT, buf, bsize, &pos, comm);
MPI_Pack(&n, 1, MPI_INT, buf, bsize, &pos, comm);
MPI_Pack(cdata, n, MPI_FLOAT, buf, bsize, &pos, comm);
MPI_Send(buf, pos, MPI_PACKED, d1, tag, comm);
MPI_Send(buf, pos, MPI_PACKED, d2, tag, comm);
MPI_Send(buf, pos, MPI_PACKED, d3, tag, comm);
```

Using the CLI, this simplifies to:

```
_Send @ [comm, destarr:nproc, tag] (pdata:nproc, cdata:n);
```

Coordination Language Interface

<op>@[<communication context>](<structured data>)

where

<op>: `_Send` **|** `_Recv`

<communication context>: <communicator>, <rank>, <tag info>

<communicator>: COMID

<rank>: RANK | RANKV:EXPR

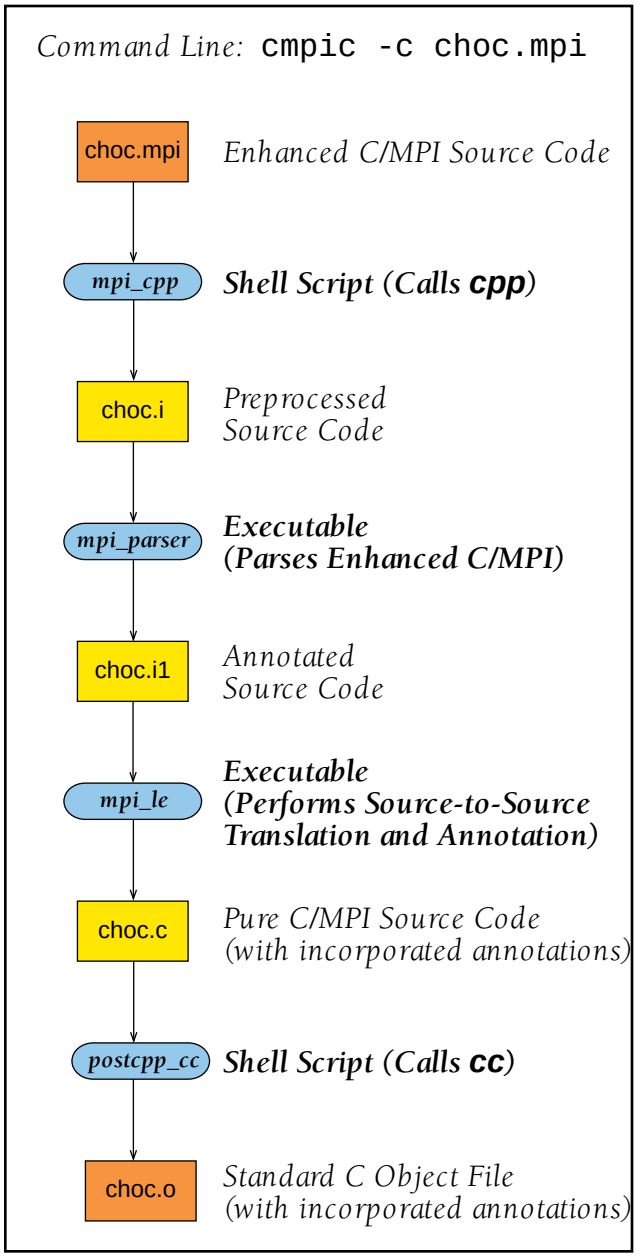
<structured data>: <datum> [, <structured data>]

<datum>: EXPR[: EXPR[:EXPR]]

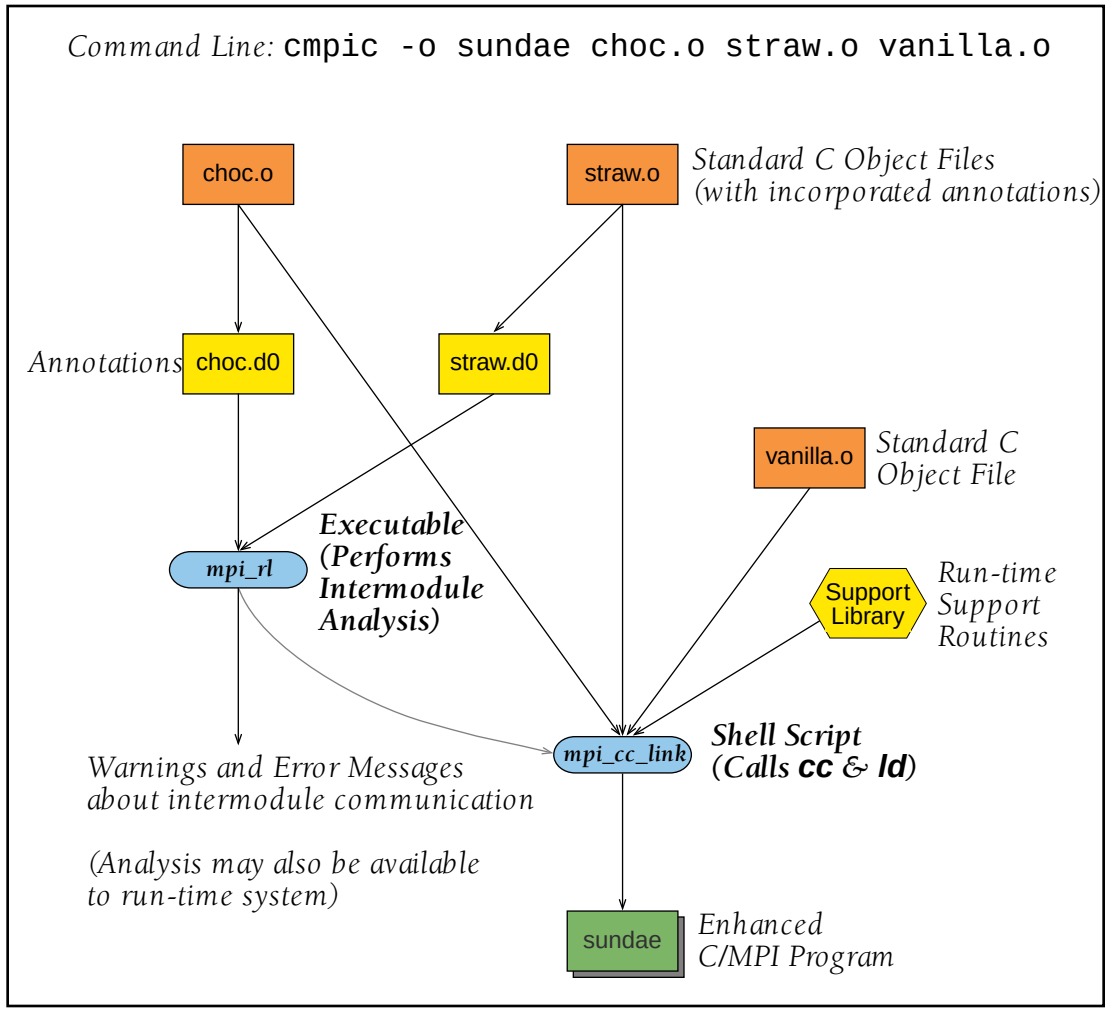
Implementation Overview

- **CLI is language-level and is processed using standard SCIENTIFIC pre-compiler and pre-linker technology:**
  - Enables substantial syntactic and semantic error checking before run time
  - Opens the potential for optimization
  - Can be easily retargetted for different message-passing systems (or for special variants like real-time)
- **Variety of possible run-time implementation designs:**
  - Permits performance vs. flexibility tradeoffs
  - Supports debugging/monitoring interfaces
  - Enables dynamic adaptation to hardware/communication information available at run time

Creating an Enhanced MPI Executable



Precompiler processing



Prelinker processing

SCIENTIFIC

Run-Time Implementations

- **Direct Replacement Source-to-Source Translation:**
  - Pre-compile/pre-link processing produces a source file containing ordinary calls to the message-passing library.
  - Linked executable uses specific message-passing routines.
  - Highly efficient (no overhead at run time).
  - Easily modifiable (source output is readable).
- **Service Routines:**
  - Pre-compile/pre-link processing replaces each CLI call with calls to generic service routines to process the communication context and the actual data motion.
  - Service routines “interpret” their arguments at run time in order to invoke proper message-passing routines.
  - Extremely flexible (service routines can be arbitrarily dynamic).
  - Ideal for tracing/debugging during development.
  - Low run-time overhead in most cases.

Adding a Virtual Shared Memory Service

- **A VSM service can be added within the message-passing semantics by using a distinguished task identifier:**

```
_Send [comm, THE_VSM, OBJ_TAG] (<structured data>);

_Recv [comm, THE_VSM, OBJ_TAG] (<structured data>);
```

- **The `OBJ_TAG` is used to identify and retrieve VSM objects.**

- **Implementation to be based on the Paradise® VSM system.**

Why Add a Virtual Shared Memory?

- **Better fit to certain algorithms:**
  - Many algorithms have data that isn't naturally "owned" by one of the processes.
  - Example: a shared counter is difficult to implement using message-passing alone, but is trivial with a VSM:

```
/* create a counter */
_Send [comm, THE_VSM, COUNTER_TAG] (0);

/* increment the counter */
_Recv [comm, THE_VSM, COUNTER_TAG] (counter);
_Send [comm, THE_VSM, COUNTER_TAG] (counter+1);
```

- **Mechanism to handle "out-of-band" data and meta-data**

Debugging/Monitoring Interfaces: Meta-Data

- **The VSM can be used to collect information about the running program and the overall system, such as:**
  - Communication state record for each process
  - Out-of-band data for each message
  - Summary communication activity tables

- **The VSM may be "open": such meta-data may be available to debuggers, monitors, performance analyzers, etc.**

- **The meta-data may also be used by applications in order to dynamically customize their communication activities.**

Status and Plans

- **Status**
  - Initial MPI prototype operational
  - Current effort aimed at extending error handling and analysis
- **Plans (Funded)**
  - Continued development compile and run-time systems for MPI
  - Hooks for tracing and debugging
  - Testing by others on real applications (looking for partners)
- **Plans (Not Yet Funded)**
  - Development of virtual shared memory service
  - Application work and tuning (in collaboration with partners)
  - Extensions to other message-passing environments (vendor-specific systems, real-time MPI, etc.)